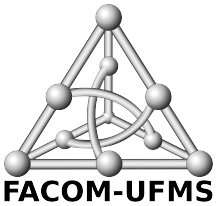
Análise da aplicação do Algoritmo Colônia de Abelhas Artificiais no problema da Mochila Multidimensional Binária

Amanda Ayumi Yamashita

Luiz Eduardo Batista Garcia

Trabalho de Conclusão de Curso

Orientação
Prof. Dra. Bianca de Almeida Dantas



Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Campo Grande - MS
2023

Resumo

A mochila multidimensional binária é um problema de otimização muito explorado no âmbito da computação. Mesmo sendo um problema conhecido, sua solução não é trivial, uma vez que pertence à classe dos problemas $\mathcal{NP}$-difíceis, isto é, não há um algoritmo que forneça uma solução que possa ser verificada em tempo polinomial. Assim, a fim de buscar soluções aproximadas em menor tempo, faz-se uso das meta-heurísticas. Neste trabalho, utilizou-se a meta-heurística do Algoritmo Colônia de Abelhas Artificiais (ABC - do inglês *Artificial Bee Colony*). Nesse algoritmo iterativo, a colônia é simplificada em três classes de abelhas: as empregadas, as observadoras e as escoteiras. Durante as iterações, elas procuram a melhor solução alcançável, que se refere à fonte de alimento procurada pela colônia. Com base nessa meta-heurística, desenvolveu-se um algoritmo ABC discreto, visto que a mochila multidimensional é binária e o ABC foi proposto inicialmente para ser aplicado a problemas contínuos, a fim de tentar alcançar bons resultados para a questão proposta. O algoritmo apresentado foi avaliado com base na resolução de problemas de otimização encontrados na literatura. Os resultados obtidos foram armazenados e processados para realizar comparações quantitativas quanto à aplicabilidade do algoritmo no problema proposto. Para comprovar a eficiência, foram calculados as médias e os desvios-padrões dos resultados, a fim de validá-los. Por fim, pode-se dizer que o algoritmo obteve resultados promissores na maioria dos casos testados, aproximando-se de outras meta-heurísticas aplicáveis existentes.

Palavras-chave: *meta-heurísticas*, *otimização combinatória*, *mochila multidimensional binária*, *algoritmo colônia de abelhas*.

CAPÍTULO 1

Introdução

Problemas de otimização estão presentes em diversos ramos das ciências e das engenharias. Consistem em encontrar a melhor solução possível ou desejável para uma questão abordada. Nesses problemas, muitas vezes haverá condições impostas, como a limitação dos recursos disponíveis, e requisitos exigidos, que devem ser atendidos da melhor forma possível. Sendo assim, os algoritmos de otimização procuram maximizar ou minimizar um ou mais elementos da função problema [4, 11].

Nesse contexto, existe o problema da mochila, KP (*Knapsack Problem*), um problema de otimização, frequentemente abordado na área da computação, e que se estende a diversas aplicações práticas no mundo real. O seu nome deriva da situação em que é necessário colocar em uma mochila objetos de diferentes pesos e valores sem exceder a sua capacidade e maximizando a soma total dos valores dos objetos.

O problema da mochila possui algumas variações, sendo elas: i) a mochila 0-1 ou mochila binária; ii) a mochila multidimensional; iii) o problema de múltiplas mochilas; e iv) a mochila fracionária. Todas as variações pertencem à classe de problemas $\mathcal{NP}$-difícil, ou seja, não é conhecido um algoritmo polinomial para sua solução exata e possuem uma ampla gama de aplicações em problemas computacionais, em programação discreta e em otimização combinatória.

Assim, um possível caminho para encontrar a solução dos problemas $\mathcal{NP}$-difíceis é por meio de heurísticas e meta-heurísticas, que podem ser vistas como ferramentas algorítmicas gerais aplicáveis a diferentes problemas de otimização, com modificações relativamente pequenas para torná-las adaptáveis a um problema específico [3].

Diversas meta-heurísticas sugeridas na literatura são inspiradas nos costumes de diversos animais na natureza. A partir dessa categoria, temos o algoritmo populacional inspirado no comportamento das abelhas [4]. Os algoritmos populacionais são aqueles que otimizam uma função objetivo, por meio da troca de informações entre indivíduos de uma população. Desse modo, a partir de um conjunto inicial de soluções, são realizadas operações até que seja encontrada uma solução ótima.

Proposto por Karaboga [8] em 2005, o Algoritmo Colônia de Abelhas Artificiais, doravante ABC (*Artificial Bee Colony*), segue conceitos baseados na ideia de inteligência coletiva dessa população, mais especificamente o conceito de forrageamento que será

abordado na Seção 2.2.

O objetivo deste trabalho é implementar e testar a heurística Algoritmo Colônia de Abelhas Artificiais, a fim de atestar a validade dessa heurística na solução do KP em sua variação multidimensional $0-1$ (MKP - *Multidimensional Knapsack Problem*), detalhada em seguida, na Seção 2.1. Assim, serão aplicados os problemas de otimização do MKP presentes nos conjunto SAC-94, GK e ORLIB.

O texto deste trabalho está organizado como segue. O Capítulo 2 descreve o problema da mochila multimensional binária e o Algoritmo Colônia de Abelhas Artificiais, que foi estudado e implementado. O Capítulo 3 apresenta o processo de implementação e discretização do algoritmo original, explicitando as escolhas de desenvolvimento e as configurações escolhidas para os parâmetros dele. Os resultados obtidos são mostrados e analisados no Capítulo 4. Por fim, o Capítulo 5 apresenta as conclusões obtidas a partir deste estudo e propostas para trabalhos futuros.

CAPÍTULO 2

Contextualização

A primeira seção do capítulo a seguir oferece uma contextualização do Problema da Mochila Multidimensional Binária, explorando a natureza e a formulação desse problema de otimização combinatória. A seção subsequente aborda o conceito e desenvolvimento do Algoritmo Colônia de Abelhas Artificiais (ABC), uma heurística inspirada no comportamento de forrageamento das abelhas.

2.1 Problema da Mochila Multidimensional Binária

A mochila binária, também conhecida como mochila 0-1, é um problema clássico de otimização combinatória, que envolve a escolha de um subconjunto de itens de um conjunto dado, de modo a maximizar o valor total desses itens, respeitando uma capacidade máxima de peso da mochila. Cada item possui um valor e um peso associados, e o objetivo é determinar quais itens devem ser colocados na mochila e quais devem ser deixados de fora, de forma que o valor total seja maximizado e o peso total não exceda a capacidade da mochila.

A formulação matemática do problema da mochila binária é a seguinte:

$$max \sum_{i=1}^{n} v_i x_i \tag{2.1}$$

tal que:

$$\sum_{i=1}^{n} p_i x_i \leq c, \ x_i \in \{0,1\} \tag{2.2}$$

Dados a capacidade c da mochila e um conjunto de n itens, em que cada item i é caracterizado por seu valor v_i e seu peso p_i, deseja-se encontrar um vetor binário $x = (x_1, x_2, ..., x_n)$, sendo x_i é a variável de decisão que demonstra se o item i está presente ($x_i = 1$) ou ausente ($x_i = 0$). Dessa forma, o objetivo é maximizar a função supracitada, isto é, maximizar a soma dos valores dos itens selecionados, respeitando a restrição da capacidade da mochila [15] [5].

A extensão multidimensional binária do problema da mochila, conhecida como *Multidimensional Knapsack Problem* (MKP), incorpora a ideia de várias dimensões de restrição, em que cada item possui um peso em cada dimensão e cada uma delas possui sua própria capacidade máxima. O objetivo é maximizar o valor total dos itens escolhidos, respeitando as restrições de peso em todas as dimensões. Esse problema pode ser modelado matematicamente, conforme a equação a seguir:

$$max \sum_{i=1}^{n} v_i x_i \tag{2.3}$$

tal que:

$$\begin{aligned} \sum_{i=1}^{n} r_{ji} x_i \leq b_j, \forall j = 1, \ldots, m \\ x_i \in \{0, 1\} \end{aligned} \tag{2.4}$$

sendo n o número de itens e m é o número de dimensões da mochila, cada dimensão com uma capacidade b_j. Cada item i requer r_{ji} recursos na dimensão j e possui um valor v_i associado, se for selecionado para a mochila. De forma análoga ao caso base, o objetivo no MKP é determinar um subconjunto de itens que maximize a soma dos valores, respeitando simultaneamente as capacidades de todas as dimensões [6].

2.2 Algoritmo Colônia de Abelhas Artificiais

O Algoritmo Colônia de Abelhas Artificiais (ABC) foi apresentado por Karaboga [8] em 2005 para a resolução de problemas de otimização multimodal e multidimensional [4]. O ABC é um algoritmo com inspiração natural que atua aplicando conceitos de inteligência coletiva apresentados por abelhas reais durante o processo de forrageamento, isto é, a busca e a exploração de recursos e fontes de alimento.

Conforme abordado por Vasconcelos et al. [13], na natureza, a atividade de forrageamento das abelhas é essencial à sua sobrevivência. Assim, uma parcela da população de abelhas operárias deve explorar o ambiente em busca de recursos florais (pólen, néctar e água), coletando informações que devem ser repassadas para as demais operárias que, por sua vez, definem qual fonte é mais rentável para a colônia. Além disso, durante o forrageamento, existe o recrutamento de abelhas para intensificar a exploração de boas fontes e o abandono de fontes esgotadas.

Em paralelo ao enxame de abelhas naturais, o ABC possui três classes de abelhas operárias, sendo elas escoteiras, empregadas e observadoras. Abelhas empregadas estão relacionadas a uma determinada fonte de alimento, isto é, uma solução candidata a resolver o problema é associada a cada abelha empregada.

Para além disso, existem as abelhas observadoras, que escolhem as fontes com base nas aptidões das soluções apontadas pelas abelhas empregadas. Por fim, as abelhas escoteiras, que irão explorar artificialmente uma nova solução candidata, caso haja falha nas soluções apontadas anteriormente.

Em síntese, o ABC pode ser descrito como um algoritmo que simula uma colônia de abelhas praticando o forrageamento na busca por uma solução ótima para um problema

de otimização [4]. O Algoritmo 2.1 descreve os principais passos para a implementação da Colônia de Abelhas Artificiais.

Algoritmo 2.1: ABC

```
1 Inicialize a população de abelhas;
2 repita
3     Posicione as abelhas empregadas em suas fontes de alimento;
4     Calcule a probabilidade de escolha da fonte por abelhas observadoras;
5     Posicione as abelhas observadoras nas fontes conforme seus valores de
      aptidão;
6     Interrompa a exploração de fontes esgotadas;
7     Envie abelhas escoteiras para buscar novas fontes de alimento;
8     Memorize a melhor fonte encontrada;
9 até Número máximo de ciclos;
```

Na etapa de inicialização do Algoritmo 1, soluções candidatas são geradas de maneira aleatória dentro do espaço de buscas. A população inicial será composta por $SN/2$ soluções, sendo SN o número total de abelhas da população. Com isso, metade do enxame conhece as fontes de alimentos (possíveis soluções) e a outra metade, composta por abelhas observadoras, escolhe de acordo com a qualidade da fonte apresentada pelas abelhas empregadas inicializadas. Em consequência disso, as fontes com altas probabilidades terão um recrutamento maior, intensificando assim a exploração de boas soluções [8, 9].

Cabe frisar que, cada abelha está associada a um vetor de solução, cuja dimensão equivale ao número de variáveis do problema abordado. Por fim, após a etapa de inicialização, o processo de busca das abelhas escoteiras, a melhora da solução das empregadas e a seleção feita pelas observadoras, repete-se num número máximo de ciclos [8, 9].

Além disso, a fim de tornar o algoritmo mais robusto e eficiente, existe a possibilidade de cada abelha empregada ou observadora alterar a fonte memorizada ao efetuar a busca local, isto é, buscar soluções na vizinhança. Assim, caso existam melhores soluções nas proximidades, as operárias devem abandonar suas fontes e explorar as mais vantajosas.

O ABC é um algoritmo que se divide em quatro processos de seleção [4], para além da inicialização da população, sendo eles:

- Seleção global na fase de abelhas observadoras (selecionam conforme função de probabilidade);
- Seleção local, ao desbravar vizinhança;
- Seleção local gulosa, ao escolher substituir ou não a solução candidata;
- Seleção aleatória na fase de abelhas escoteiras.

CAPÍTULO 3

Implementação

Conforme posto no Capítulo 2, o MKP é um problema de otimização binário, enquanto o ABC é projetado para otimizar problemas contínuos. Logo, faz-se necessário discretizar o algoritmo descrito por Karaboga [8]. O funcionamento do ABC binário proposto não se diferencia com relação ao Algoritmo 2.1. No entanto, algumas funções diferem das apresentadas pelo autor.

O enxame é composto por abelhas artificiais contendo uma solução candidata, ou seja, um vetor binário de dimensão n, em que n é o número de itens apresentado no KP. Todas as abelhas possuem um componente peso, que equivale ao peso da mochila para a solução apresentada. Por fim, uma variável de ciclo que assinala o estágio de exploração da fonte, representada por um valor númerico entre $\{0, 1, ...,$limite ciclo$\}$.

A inicialização da população foi feita de forma aleatória, sendo que cada variável de decisão é o número 0, caso o item não esteja presente na solução, ou 1, caso esteja. Porém, tanto a população inicial, quanto as remanescentes são compostas apenas por indivíduos viáveis, ou seja, abelhas associadas a soluções que não ultrapassem as restrições de recursos da mochila. Para garantir a viabilidade das soluções, adotou-se uma política de correção da solução.

Com esse intuito, logo após a geração da fonte, verifica-se se ela satisfaz as restrições do problema e, caso contrário, a correção é realizada ao se retirar um item aleatório (*drop*) até que a solução se torne viável. Esta operação é essencial para o funcionamento do programa, pois, isso permite que o enxame não explore fontes inviáveis, mas sim aquelas que respeitem os limites predeterminados.

Após a inicialização, as abelhas empregadas são incumbidas de buscar fontes mais vantajosas na vizinhança. Para tal fim, aplica-se a equação:

$$u_i = x_i + \Phi(x_i - x_j) \tag{3.1}$$

em que u_i é um valor candidato para substituir x_i, $\{i, j\} \in \{1,2,...,n\}$ são índices escolhidos aleatoriamente, $i \neq j$ e Φ é um valor aleatório entre [-1,1] [15]. Diferentemente da solução proposta originalmente por Karaboga [8], o vizinho foi tomado com base na própria solução, isto é, uma nova versão da solução original é criada baseada na alteração da posição x_i do vetor solução, utilizando a Equação (3.1), na qual x_i

e x_j são diferentes posições do vetor solução original. Para além disso, adotou-se 10 como o limite de alterações na própria solução, que será o limite local.

A variável ajustada u_i da Equação (3.1) é um número inteiro. Assim, aplica-se a Equação (3.2) para normalizar o resultado decidindo entre 0 e 1:

$$u_i = \begin{cases} 1, se\ u_i >= 1; \\ 0, se\ u_i <= 0. \end{cases} \tag{3.2}$$

Após a inicialização da população e a busca na vizinhança, abelhas observadoras irão explorar uma fonte artificial conforme as taxas de desvio respectivamente apresentadas pelas soluções, o que garante que as fontes consideradas mais eficientes para o problema terão um recrutamento maior e consequentemente a melhor solução será mais explorada e aperfeiçoada. A taxa de desvio da solução é calculada por:

$$d_i = \frac{valorOtimo - valorAbelha}{valorOtimo} \tag{3.3}$$

Sendo *valorOtimo* a solução ótima conhecida apresentada pelo conjunto de testes e *valorAbelha* a solução encontrada pela abelha no ciclo de execução, ou seja, o peso da abelha. Em resumo, abelhas observadoras irão explorar as melhores fontes conforme a Equação (3.3) e as abelhas empregadas irão explorar suas respectivas fontes.

Já as abelhas escoteiras são aquelas que não obtiveram sucesso nas fases anteriores. Assim sendo, abelhas empregadas e observadoras podem se tornar escoteiras dado o limite de ciclos estabelecidos para a variável de ciclo. Desse modo, soluções esgotadas não consomem recursos indefinitivamente e as abelhas escoteiras buscam novas fontes para explorar.

O funcionamento do algoritmo ocorre por meio de ciclos globais, em que abelhas observadoras e escoteiras exploram suas fontes. O limite global se refere ao número máximo de ciclos no qual as soluções criadas na inicialização podem ser melhoradas, ou seja, após a inicialização das abelhas, elas terão até o limite global de vezes para encontrar a solução ótima do teste fornecido, caso contrário, o resultado alcançado é gravado, e continua-se para a próxima execução, como pode ser visto no diagrama apresentado na Figura 3.1, o ciclo mais externo representa o limite global.

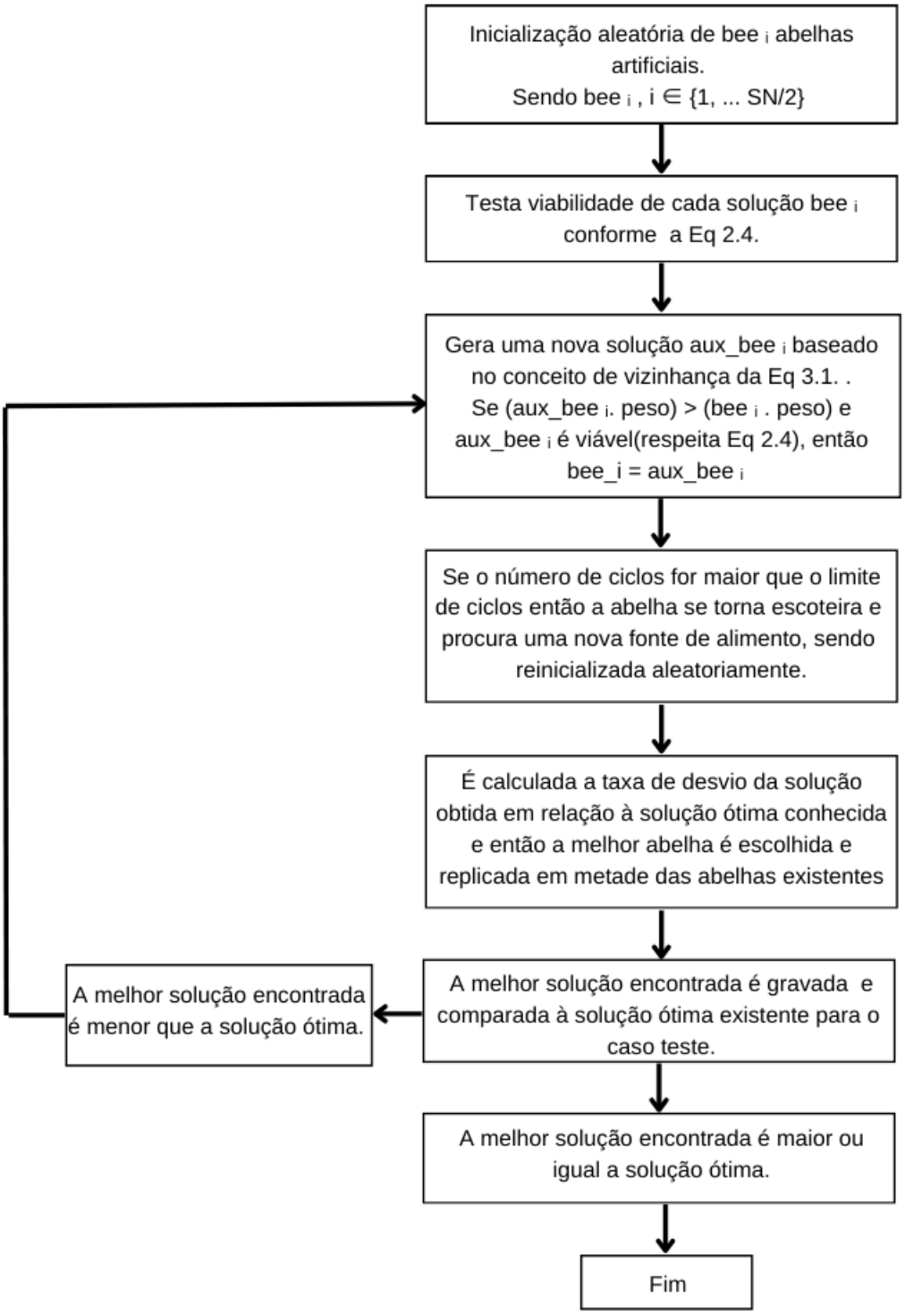


Figura 3.1: Diagrama demonstrativo do funcionamento do código.

Em concordância a outros trabalhos [4, 12], corrobora-se a ideia de que com o aumento do número de iterações do algoritmo, existe a possibilidade das abelhas artificiais explorarem fontes muito próximas ou até mesmo as mesmas durante o forrageamento. Dada a relevância dessa característica e somado ao fato que as variáveis são binárias , optou-se por um limite local pequeno. Em contrapartida, obteve-se melhores resultados ao adotar o limite global maior.

Diante do exposto, os parâmetros de configuração adotados foram os seguintes:

- População de abelhas = 100;
- Limite vizinhança = 10;
- Limite local = 10;
- Limite global = 1000, 5000 e 10000.

Essa variação no limite global foi adotada a fim de identificar possíveis padrões de comportamento das abelhas, visto que, apesar da inicialização aleatória foi observado um afunilamento nas soluções candidatas. Além disso, foi possível comparar o desempenho da colônia artificial para cada limite.

Após a execução completa do algoritmo, uma solução é proposta para o problema da mochila com base na melhor abelha. Para medir a qualidade da solução apresentada, utiliza-se o conceito de *gap*, que representa a porcentagem da diferença da melhor solução conhecida e da solução obtida e é calculada com o uso da Equação (3.4):

$$gap = 100 * (valorOtimo - valorObtido)/valorOtimo \tag{3.4}$$

O *gap* é a representação em porcentagem da Equação (3.3). Os valores de referência *valorOtimo* são as melhores soluções conhecidas pela literatura até o momento e *valorAbelha* a solução encontrada pela melhor abelha da execução. A fim de mensurar a robustez do sistema, repetiu-se todo o processo descrito no algoritmo ABC discreto 30 vezes consecutivas.

CAPÍTULO 4

Análise de Resultados

Para a avaliação do algoritmo proposto neste trabalho, três diferentes conjuntos de testes encontrados na literatura foram usados: o conjunto SAC-94, a biblioteca ORLIB e o conjunto GK.

As soluções de referência são os melhores valores conhecidos para as instâncias de SAC-94 e de OR até o momento desta pesquisa. Muitos desses valores são reconhecidos como soluções ótimas, tendo sido previamente comprovados como tal. No caso das instâncias de GK, foram utilizados os valores obtidos pelo CPLEX, como apresentados em [5].

Os experimentos foram conduzidos em uma máquina com a seguinte configuração:

- Processador Intel Core 2 Duo E8400 de 2 núcleos e 3GHz de frequência;
- 3.8 GB de memória;
- 250 GB de capacidade de disco;
- Gráficos Ilvmpipe (LLVM 12.0.0, 128 bits).

Essa configuração foi utilizada para garantir um ambiente de testes consistente e replicável, permitindo uma análise precisa do desempenho do algoritmo proposto. Além disso, a execução do ABC foi repetida 30 vezes para cada instância de teste, a fim de encontrar um valor médio para os resultados aqui descritos.

4.1 Resultados Obtidos

O primeiro conjunto foi o da biblioteca SAC-94 que é composta por 6 conjuntos e 54 problemas ao todo. Esses conjuntos são baseados em problemas reais apresentados em artigos científicos, contendo instâncias entre 10 e 105 itens e de 2 a 30 recursos, representados por n e m, respectivamente. Os resultados obtidos com as execuções dos testes para esta biblioteca podem ser vistos na Tabela 4.1.

Tabela 4.1: Resultados do Algoritmo ABC para as instâncias de SAC-94.

Conjunto	Instâncias	Iterações	Ótimos	*Gap* Mínimo	*Gap* Médio	Tempo Médio
hp	2	1000	0	1.4043	2.2372	0.0844
		5000	8	0.0000	1.0583	0.3443
		10000	20	0.0000	0.5327	0.5289
pb	6	1000	0	1.0628	4.1727	0.1670
		5000	51	0.0000	1.9704	0.7237
		10000	70	0.0000	1.3757	1.3565
pet	6	1000	68	0.0000	0.7969	0.0721
		5000	106	0.0000	0.3576	0.2269
		10000	119	0.0000	0.2797	0.4059
sento	2	1000	0	2.0523	8.6048	0.4745
		5000	0	0.8140	4.6845	2.3606
		10000	0	0.1801	3.1910	4.7131
weing	8	1000	0	0.0638	4.3709	0.0946
		5000	25	0.0000	2.2098	0.4190
		10000	106	0.0000	1.4018	0.6865
weish	30	1000	1	0.0000	7.4641	0.1644
		5000	155	0.0000	2.7658	0.7566
		10000	194	0.0000	1.8490	1.4294

Conforme visto na Tabela 4.1, o Algoritmo ABC demonstrou uma notável eficácia na resolução dos problemas da biblioteca SAC, produzindo diversas soluções ótimas equivalentes às de referência para as instâncias testadas. Além disso, observou-se uma relação direta entre o número de iterações com o tempo médio de execução, ou seja, ao aumentar o número de iterações da população, o tempo de execução do programa também se elevou. No que se refere ao *gap* mínimo e *gap* médio a situação se inverteu. Em geral, observou-se uma considerável redução nesses valores, o que está alinhado com o objetivo principal de minimizá-los.

Quando se trata das variações no número de iterações globais da população de abelhas, foi observado um bom desempenho com limites de 5000 e 10000 iterações, nos quais foram encontradas diversas soluções ótimas. Notavelmente o tempo médio praticamente dobra ao definir os limites em 5000 e 10000 ciclos. Nesse contexto, há uma variação de tempo de 0.1846, 0.6328, 0.179, 2.3525, 0.2675 e 0.6728 segundos para hp, pb, pet, sento, weing e weish, respectivamente.

O segundo conjunto testado foi o da biblioteca GK, proposta por Glover e Kochenberger [7]. O GK é composto por 11 conjuntos de testes, contendo instâncias com número de itens (n) de 100, 150, 200, 500, 1500 e 2500 e número de recursos (m) variando de 15, 25, 50 e 100. Os resultados obtidos para os conjuntos GK podem ser vistos na Tabela 4.2.

Tabela 4.2: Resultados do Algoritmo ABC para as instâncias de GK.

Conjunto	m	n	Iterações	Ótimos	*Gap* Mínimo	*Gap* Médio	Tempo Médio
gk01	15	100	1000	0	2.8943	3.5599	0.3866
			5000	0	2.3101	2.8616	1.9185
			10000	0	2.0446	2.7058	3.8185
gk02	25	100	1000	0	2.5265	3.4883	0.5789
			5000	0	2.1981	2.8432	2.8859
			10000	0	1.6928	2.5274	5.7544
gk03	25	150	1000	0	2.6344	3.4435	0.8641
			5000	0	1.9979	2.9438	4.2872
			10000	0	2.1924	2.6927	8.6340
gk04	50	150	1000	0	2.7224	3.1409	1.5938
			5000	0	2.2369	2.7484	7.9505
			10000	0	2.0288	2.5033	15.8987
gk05	25	200	1000	0	3.1229	3.4718	1.1487
			5000	0	2.5142	2.8839	5.7098
			10000	0	2.1172	2.7237	11.4543
gk06	50	200	1000	0	2.1507	2.8706	2.1413
			5000	0	2.0594	2.4635	10.7084
			10000	0	1.9030	2.3571	21.4425
gk07	25	500	1000	0	3.5285	3.6987	2.9508
			5000	0	2.8467	3.2724	14.7260
			10000	0	2.8623	3.1892	29.4767
gk08	50	500	1000	0	2.6534	2.8968	5.5448
			5000	0	2.3716	2.6132	27.7094
			10000	0	2.3131	2.5699	55.3424
gk09	25	1500	1000	0	3.4673	3.6035	9.2198
			5000	0	3.2005	3.3984	45.9224
			10000	0	3.0817	3.3135	91.8545
gk10	50	1500	1000	0	2.5936	2.7358	17.4324
			5000	0	2.4627	2.6071	86.7239
			10000	0	2.3563	2.5641	173.3848
gk11	100	2500	1000	0	2.0540	2.1466	57.4251
			5000	0	1.9720	2.0360	286.5257
			10000	0	1.9059	2.0158	572.5875

Assim como verificado para a biblioteca SAC-94, os testes evidenciam uma melhora de desempenho ao aumentar o número de ciclos globais empregado ao ABC. À medida que a escala dos ciclos aumenta, os valores de *gap* mínimo e médio são menores, sendo de interesse *gap* valendo 0 ou próximo de 0. No entanto, essa otimização não vem sem custo, uma vez que implica um aumento significativo no tempo de execução do algoritmo.

Entretanto, em contraste com os resultados obtidos na biblioteca SAC-94, não foram alcançadas as soluções ótimas conhecidas nas configurações que adotamos para o ABC. Logo, foram obtidos resultados inferiores para todos os conjuntos GK em comparação aos conjuntos SAC.

Por fim, o último conjunto de testes realizados foi na biblioteca ORLIB, proposta por Chu e Beasley [2]. Os programas desenvolvidos foram executados utilizando como entrada o conjunto de testes da biblioteca OR. A biblioteca é composta de instâncias

de problemas com 5, 10 ou 30 recursos e 100, 250 ou 500 itens.

Para melhor visualização, optou-se por separar os resultados tabulados em 3 partes distintas. Sendo eles para ORLIB com 5 restrições (OR5), 10 restrições (OR10) e 30 restrições (OR30), dispostos nas tabelas Tabela 4.3, Tabela 4.4 e Tabela 4.5, respectivamente.

Vale ressaltar que, nas tabelas a seguir o valor α diz respeito ao "fator de aperto" (*tightness factor*), que determina como os valores do vetor de recursos disponíveis, n, se relacionam com a demanda por tais recursos e pode assumir os valor 0,25, 0,50 ou 0,75, isto é, a instância do problema se torna mais restrita quando α tende a zero [5].

Tabela 4.3: Resultados do Algoritmo ABC para as instâncias de ORLIB com 5 restrições (OR5).

m	n	α	Iterações	Ótimos	*Gap* Mínimo	*Gap* Médio	Tempo Médio
5	100	0.25	1000	0	7.9460	12.5114	0.2461
			5000	0	4.1302	8.9857	1.2759
			10000	0	3.2792	7.4480	2.4329
		0.50	1000	0	5.4331	10.3687	0.2208
			5000	0	2.5114	7.1972	1.1447
			10000	0	1.4903	5.8378	2.1785
		0.75	1000	0	2.3049	8.3310	0.2116
			5000	0	0.9084	5.7028	1.1026
			10000	0	0.6259	4.5945	2.0902
	250	0.25	1000	0	12.5281	10.3845	0.3387
			5000	0	8.9335	7.5073	1.7152
			10000	0	7.5482	6.1990	3.3432
		0.50	1000	0	7.9281	10.3955	0.3616
			5000	0	5.6313	7.5865	1.8166
			10000	0	4.1518	6.2450	3.5710
		0.75	1000	0	4.2000	9.5833	0.3753
			5000	0	2.2367	6.9980	1.8753
			10000	0	1.8492	5.7508	3.7020
	500	0.25	1000	0	15.7289	10.8761	0.5888
			5000	0	11.6378	8.1887	2.9277
			10000	0	9.6997	6.8312	5.8194
		0.50	1000	0	10.0971	11.0198	0.6271
			5000	0	7.3379	8.3824	3.1125
			10000	0	6.5226	7.0131	6.2029
		0.75	1000	0	4.8123	10.5000	0.6516
			5000	0	3.8377	8.0094	3.2306
			10000	0	3.0791	6.7066	6.4492

Tabela 4.4: Resultados do Algoritmo ABC para as instâncias de ORLIB com 10 restrições (OR10).

m	n	α	Iterações	Ótimos	*Gap* Mínimo	*Gap* Médio	Tempo Médio
10	100	0.25	1000	0	8.0428	10.7502	0.6225
			5000	0	4.7724	8.1236	3.0902
			10000	0	3.4903	6.8165	6.1858
		0.50	1000	0	4.8505	10.5481	0.5935
			5000	0	2.4159	7.9067	2.9490
			10000	0	1.9269	6.6086	5.9186
		0.75	1000	0	1.9577	10.0445	0.5694
			5000	0	1.1494	7.4817	2.8319
			10000	0	0.4786	6.2431	5.6994
	250	0.25	1000	0	12.2206	10.5254	0.6036
			5000	0	7.9623	7.8538	3.0024
			10000	0	6.3382	6.5614	6.0783
		0.50	1000	0	8.1650	10.5376	0.6125
			5000	0	5.0726	7.8656	3.0466
			10000	0	4.2340	6.5562	6.2024
		0.75	1000	0	4.0274	10.2104	0.6196
			5000	0	2.3366	7.6143	3.0820
			10000	0	1.4542	6.3372	6.3087
	500	0.25	1000	0	13.3793	10.6882	0.7362
			5000	0	10.6655	8.0526	3.6613
			10000	0	9.6197	6.7506	7.5389
		0.50	1000	0	10.3767	10.7731	0.7791
			5000	0	7.7899	8.1527	3.8760
			10000	0	6.8737	6.8487	8.0342
		0.75	1000	0	5.4085	10.5333	0.8136
			5000	0	3.7633	7.9834	4.0484
			10000	0	3.0514	6.7063	8.4485

Tabela 4.5: Resultados do Algoritmo ABC para as instâncias de ORLIB com 30 restrições (OR30).

m	n	α	Iterações	Ótimos	*Gap* Mínimo	*Gap* Médio	Tempo Médio
30	100	0.25	1000	0	6.3327	10.6496	0.8130
			5000	0	3.3400	8.0122	4.0464
			10000	0	2.8785	6.7148	8.4226
		0.50	1000	0	4.9861	10.5532	0.8083
			5000	0	2.1979	7.8872	4.0231
			10000	0	1.3097	6.5919	8.3581
		0.75	1000	0	1.9765	10.2770	0.8052
			5000	0	1.1972	7.6491	4.0076
			10000	0	0.7496	6.3802	8.3113
	250	0.25	1000	0	12.6368	10.5478	0.8670
			5000	0	7.5666	7.8503	4.3175
			10000	0	5.7727	6.5433	8.9101
		0.50	1000	0	8.1066	10.5668	0.9085
			5000	0	5.1296	7.8572	4.5254
			10000	0	4.0346	6.5404	9.3094
		0.75	1000	0	4.5057	10.3772	0.9476
			5000	0	2.6609	7.7063	4.7209
			10000	0	1.8724	6.4113	9.6857
	500	0.25	1000	0	14.8268	10.6758	1.1060
			5000	0	10.1330	7.9708	5.5091
			10000	0	8.0777	6.6403	11.2457
		0.50	1000	0	10.6261	10.7448	1.2079
			5000	0	7.7696	8.0454	6.0180
			10000	0	6.2187	6.7108	12.2493
		0.75	1000	0	5.4433	10.5927	1.2993
			5000	0	3.5037	7.9401	6.4760
			10000	0	3.2625	6.6254	13.1511

O comportamento do Algoritmo ABC ocorreu, para as instâncias do OR, de maneira similar ao constatado nas bibliotecas SAC-94 e GK, ou seja, o desempenho e o tempo de execução cresceram com o número de iterações, já o *gap* mínimo e médio diminuíram conforme o número de iterações é elevado.. Nenhum dos testes no conjunto OR alcançou o valor de referência ótimo, ainda assim apresentaram *gaps* mínimos evidentemente melhores que na biblioteca GK, isto é, mais próximos de 0.

Logo, conforme observado em todos os conjuntos de testes, pode-se concluir que quanto maior o número de iterações, mais robusto é o algoritmo, minimizando o *gap* e aumentando o número de soluções ótimas, ou próximas das ótimas encontradas. Além disso, conforme esperado, quanto mais iterações a população executa, aumenta-se também o tempo de execução do algoritmo, isso ocorre devido ao efeito do tempo de exploração das abelhas artificiais, que quanto maior é, mais exploradas e otimizadas as soluções candidatas são. Vale ressaltar que houve casos significativos desse aumento, como pode ser visto no exemplo da instância gk11, mencionada anteriormente.

À medida que as dimensões do problema aumentam para cada conjunto de teste (SAC-94, GK e OR), o comportamento do ABC se altera conforme as observações feitas anteriormente. O desempenho, isto é, o *gap*, pode ser notado pelos gráficos nas

Figuras 4.1, 4.3, 4.5, 4.7, 4.9, 4.11, 4.13, 4.15, 4.17, 4.19 e 4.21. Enquanto os tempos de execução estão expostos nas Figuras 4.2, 4.4, 4.6, 4.8, 4.10, 4.12, 4.14, 4.16, 4.18, 4.20 e 4.22.

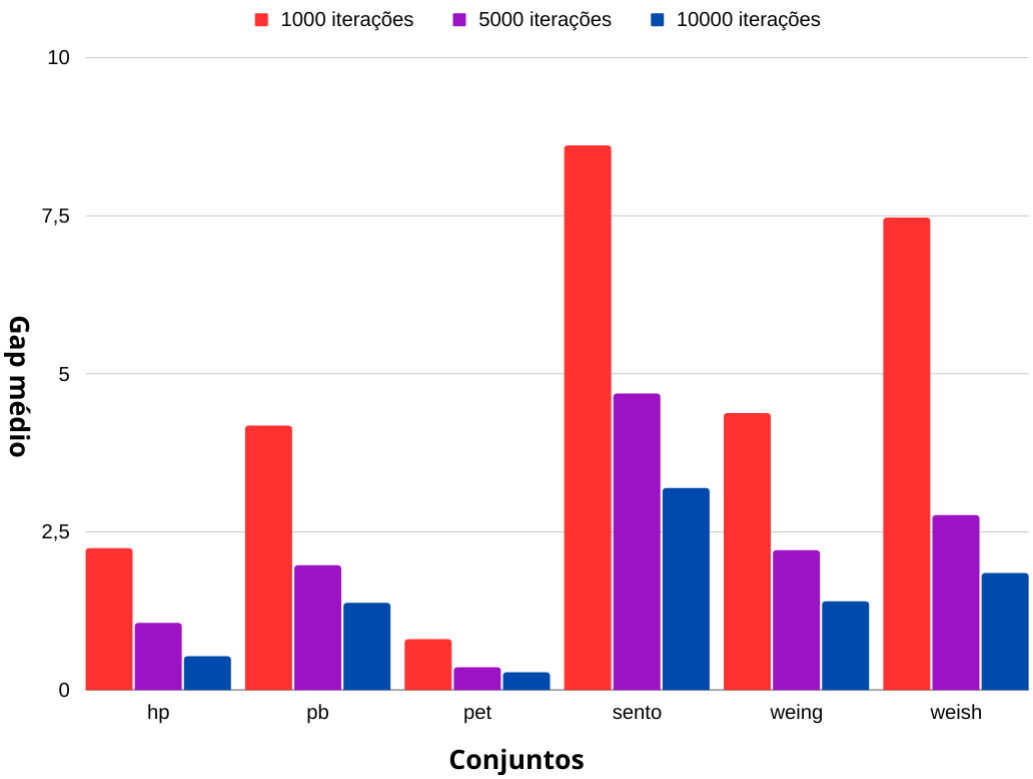


Figura 4.1: *Gaps* obtidos pelas instâncias SAC-94

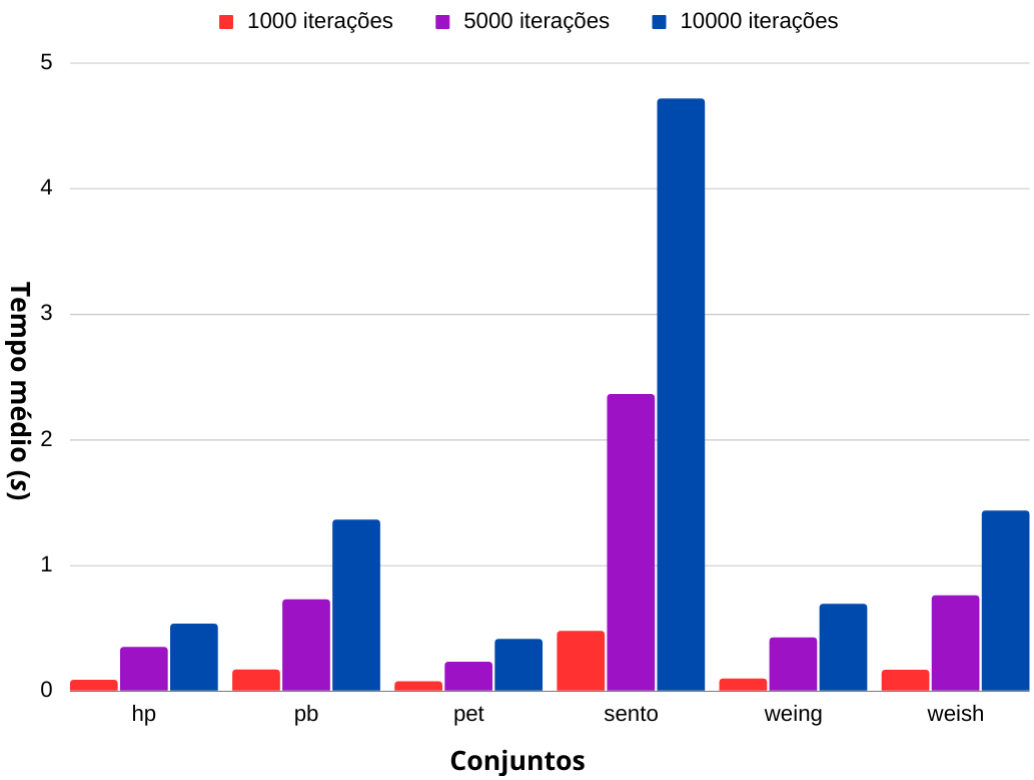


Figura 4.2: Tempos de execução obtidos pelas instâncias SAC-94

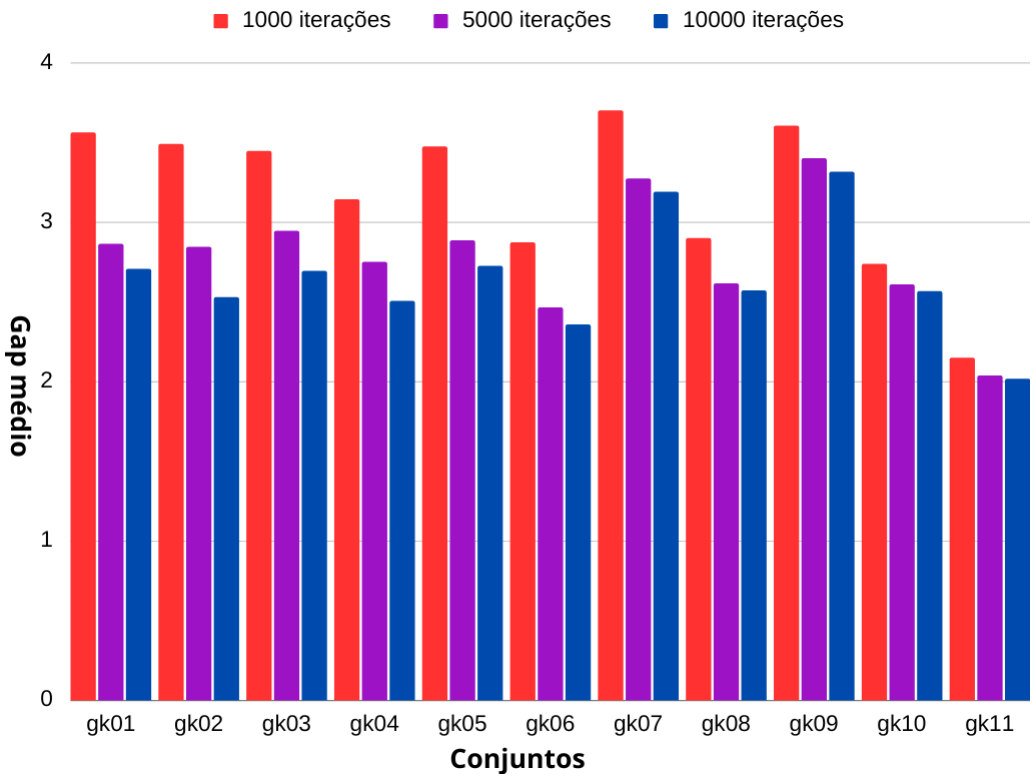


Figura 4.3: *Gaps* obtidos pelas instâncias GK

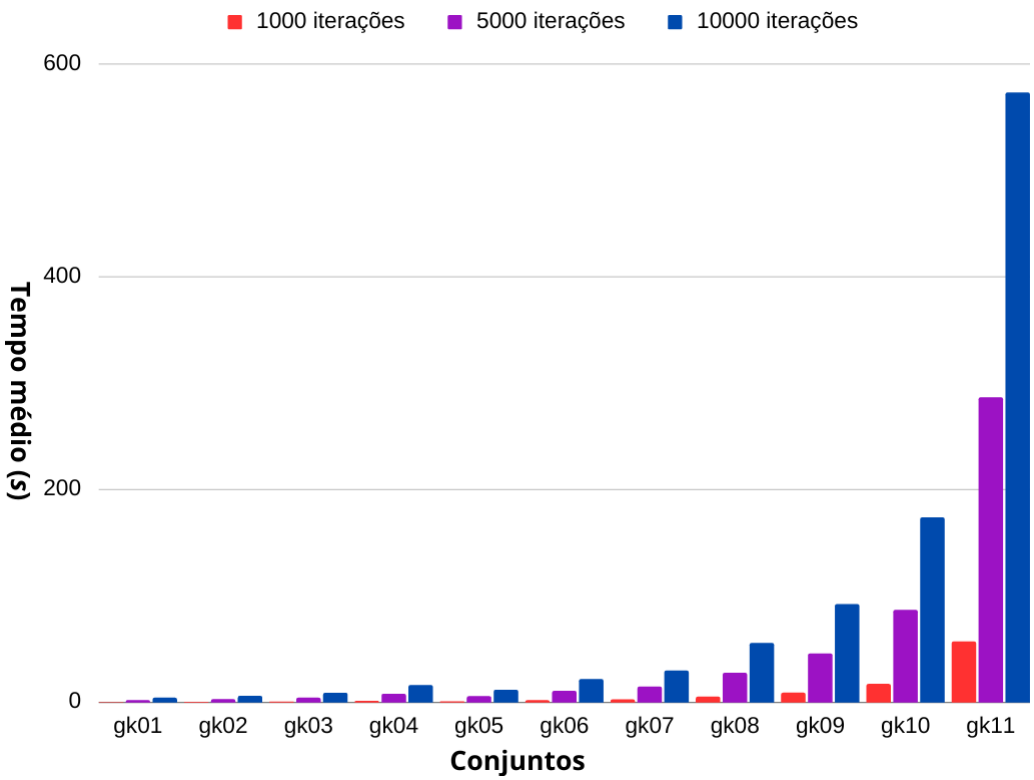


Figura 4.4: Tempos de execução obtidos pelas instâncias GK

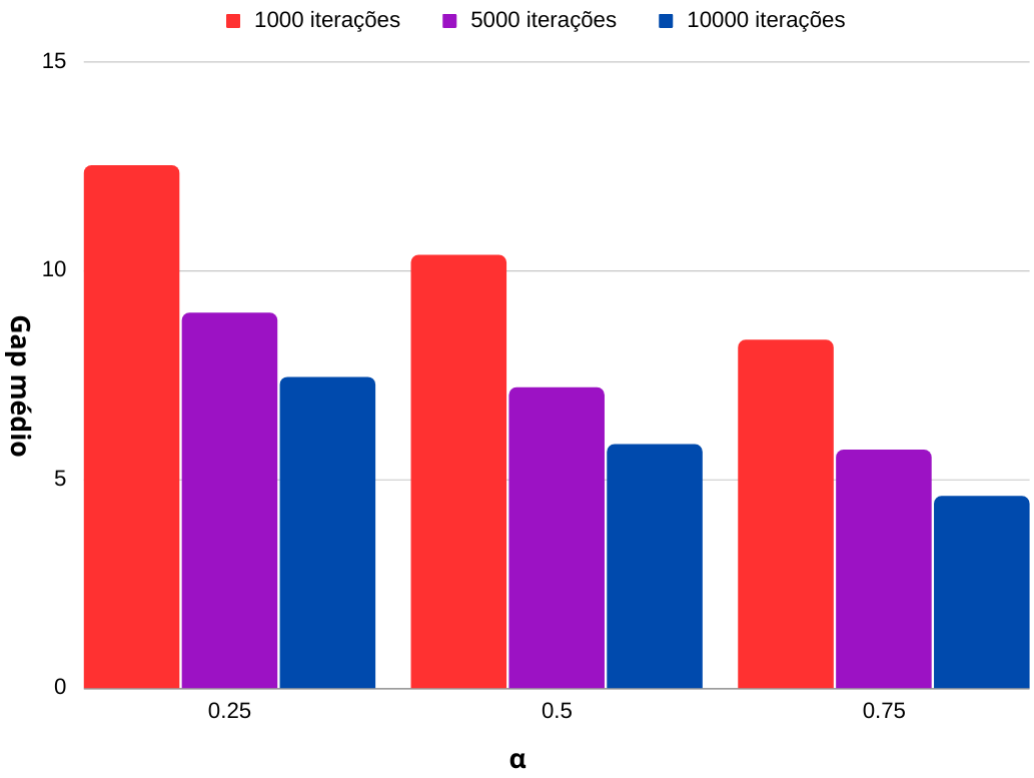


Figura 4.5: *Gaps* obtidos para as instâncias com 5 recursos e 100 itens de ORLIB

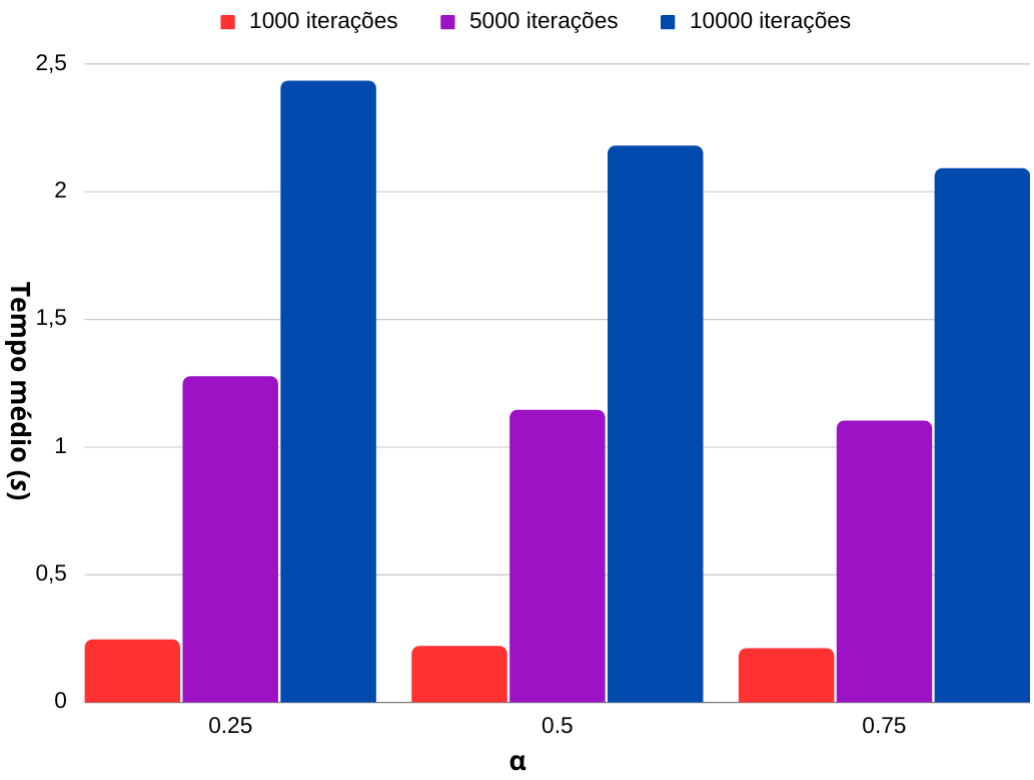


Figura 4.6: Tempos de execução obtidos para as instâncias com 5 recursos e 100 itens de ORLIB

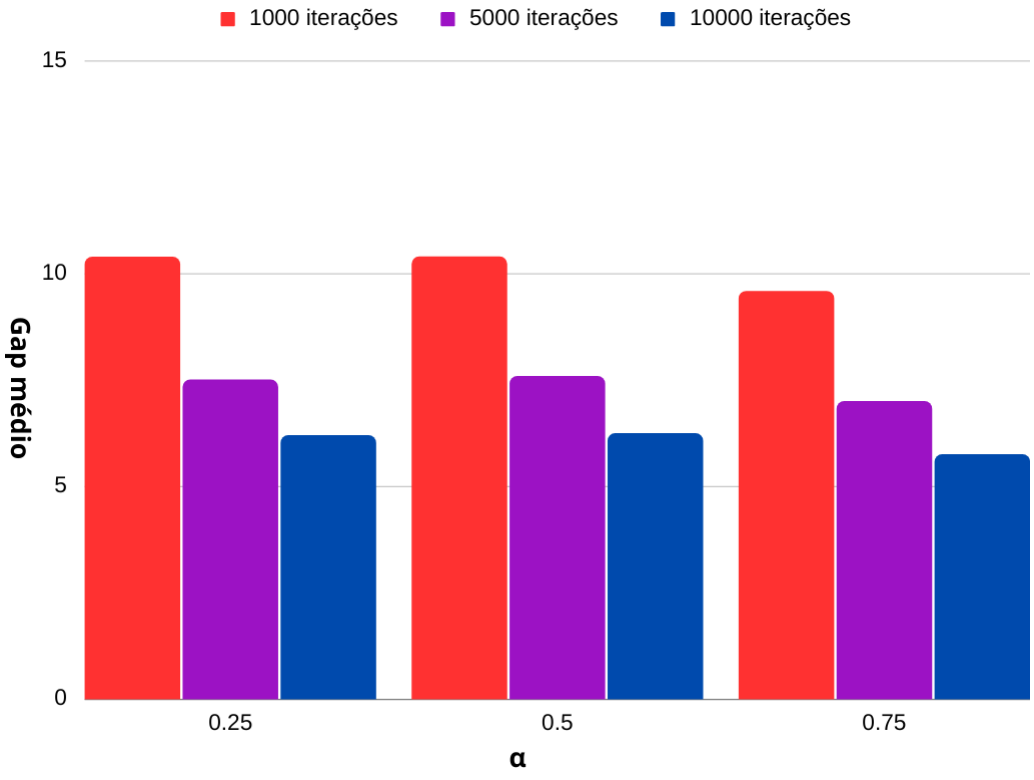


Figura 4.7: *Gaps* obtidos para as instâncias com 5 recursos e 250 itens de ORLIB

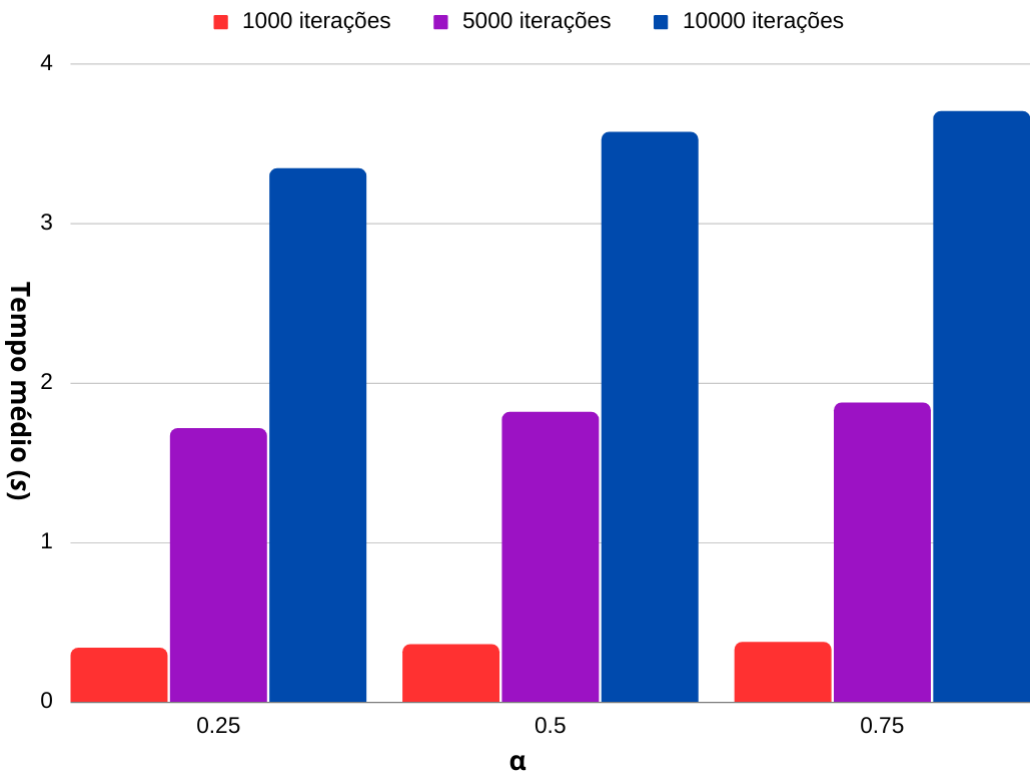


Figura 4.8: Tempos de execução obtidos para as instâncias com 5 recursos e 250 itens de ORLIB

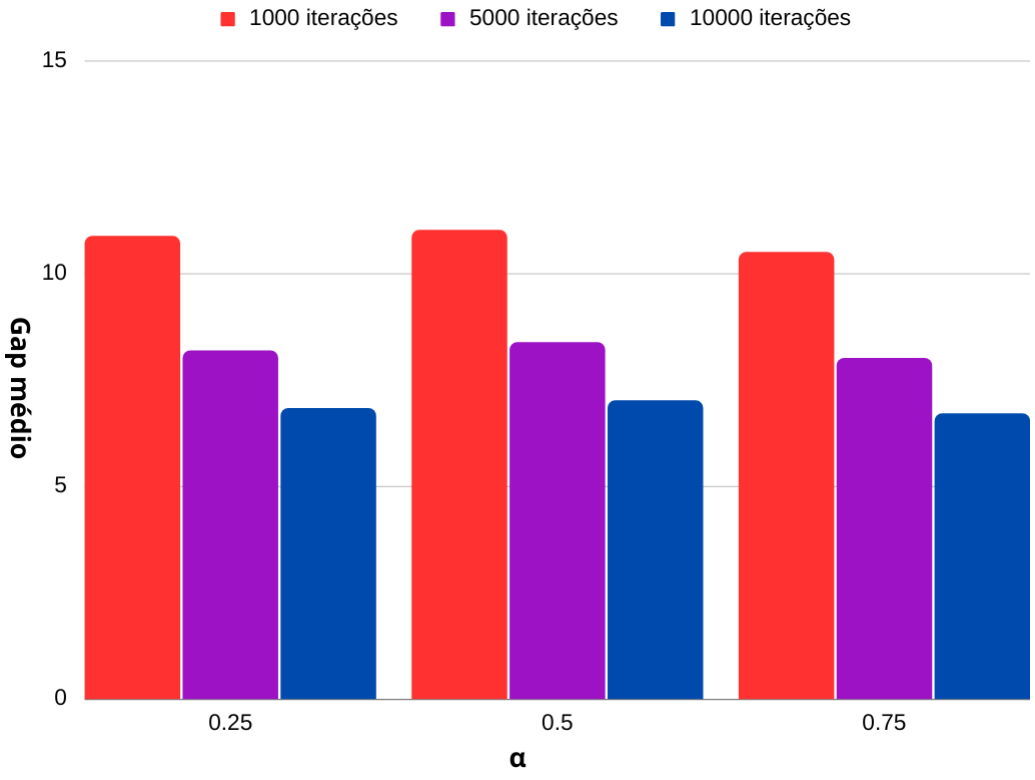


Figura 4.9: *Gaps* obtidos para as instâncias com 5 recursos e 500 itens de ORLIB

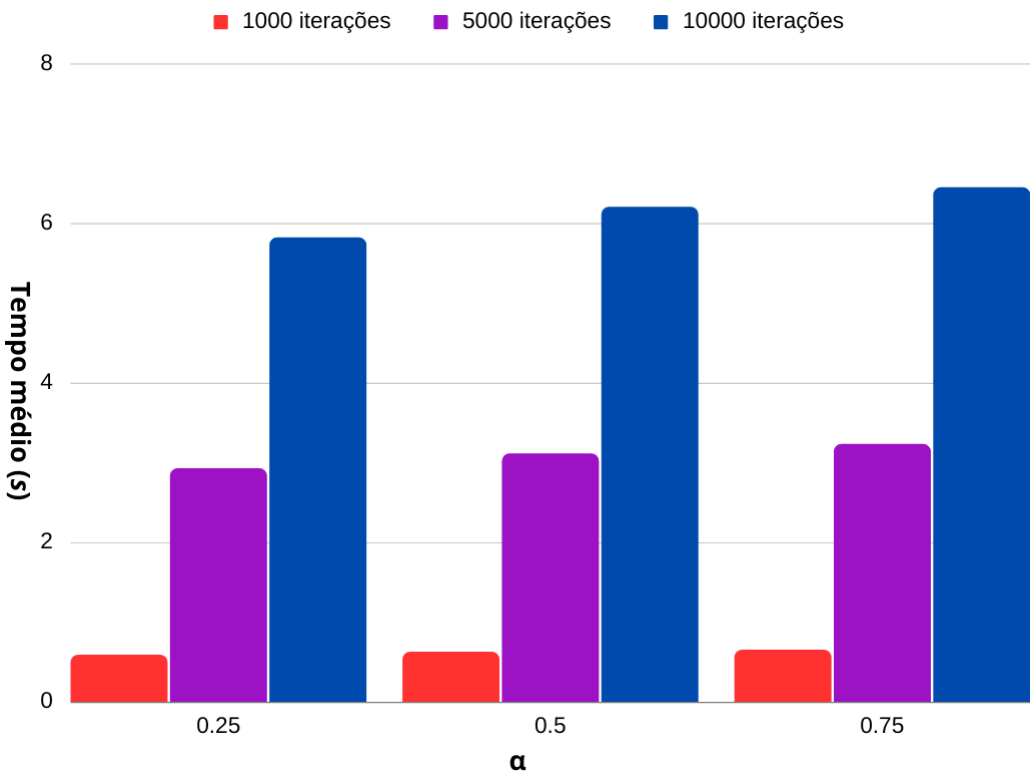


Figura 4.10: Tempos de execução obtidos para as instâncias com 5 recursos e 500 itens de ORLIB

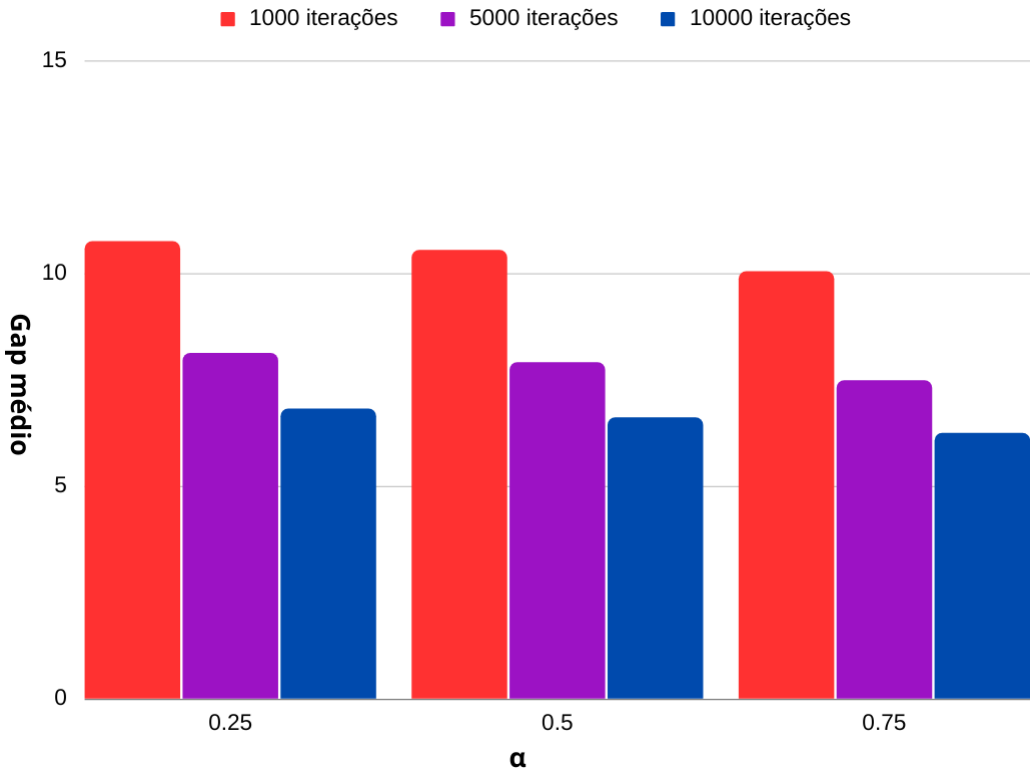


Figura 4.11: *Gaps* obtidos para as instâncias com 10 recursos e 100 itens de ORLIB

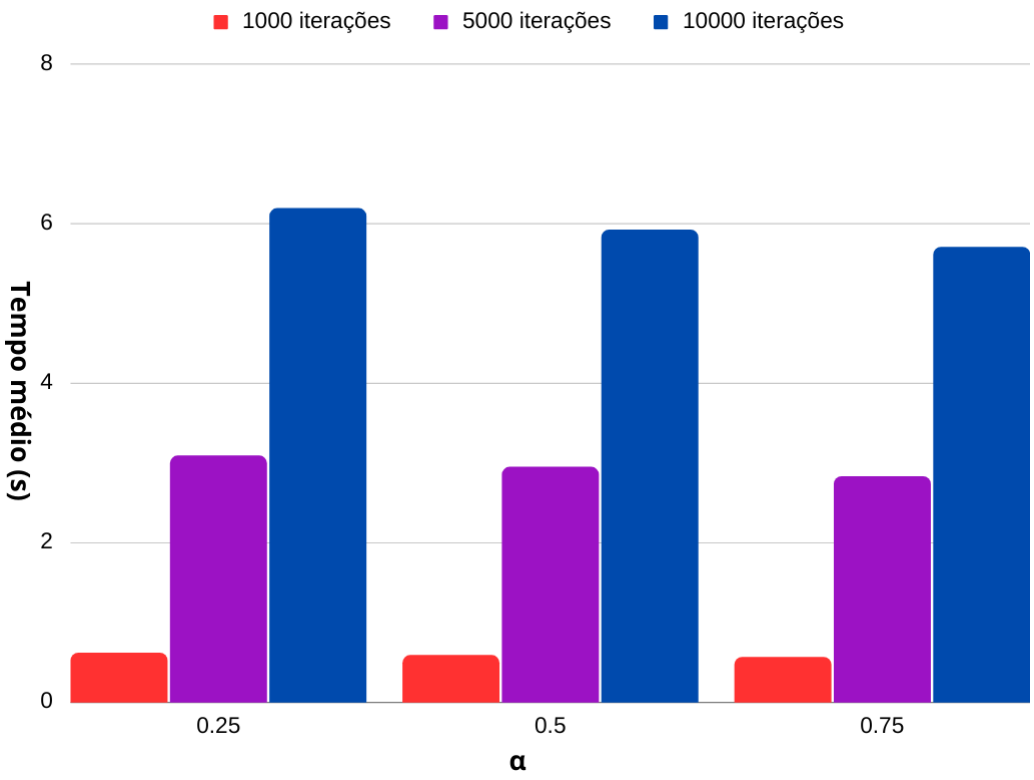


Figura 4.12: Tempos de execução obtidos para as instâncias com 10 recursos e 100 itens de ORLIB

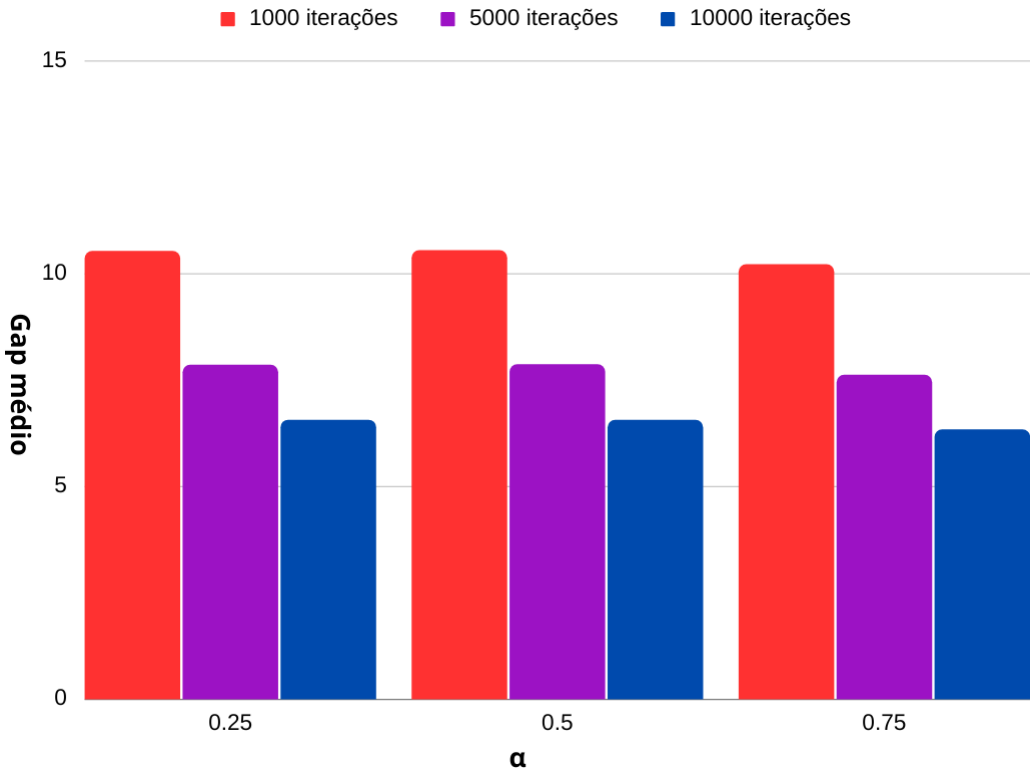


Figura 4.13: *Gaps* obtidos para as instâncias com 10 recursos e 250 itens de ORLIB

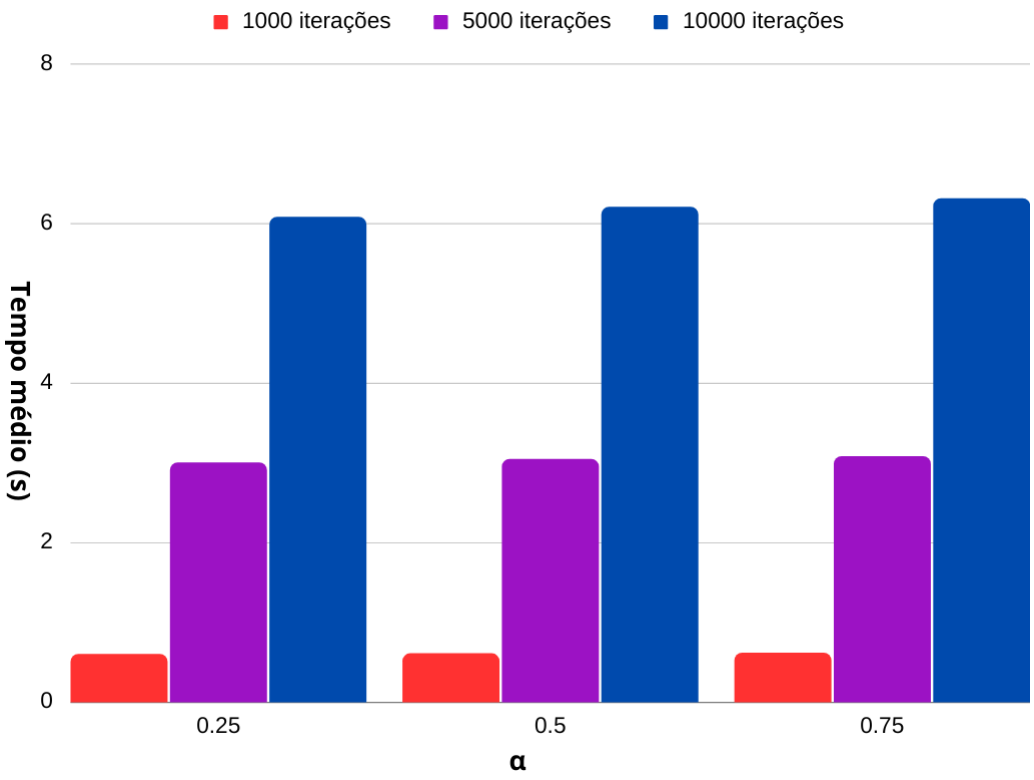


Figura 4.14: Tempos de execução obtidos para as instâncias com 10 recursos e 250 itens de ORLIB

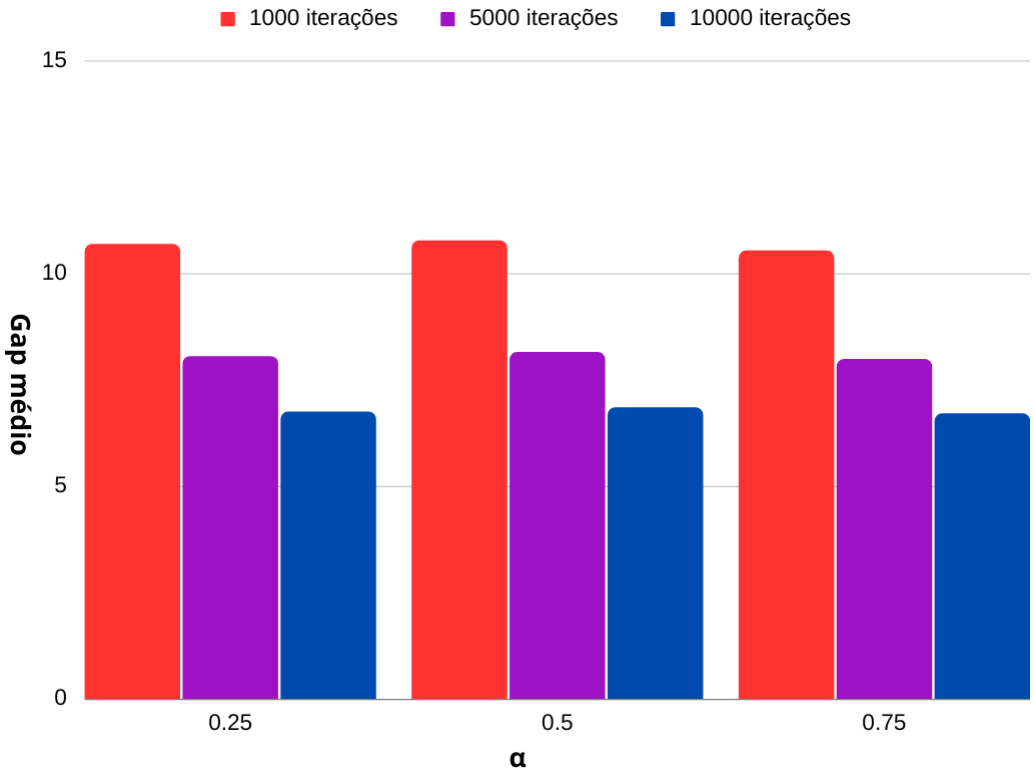


Figura 4.15: *Gaps* obtidos para as instâncias com 10 recursos e 500 itens de ORLIB

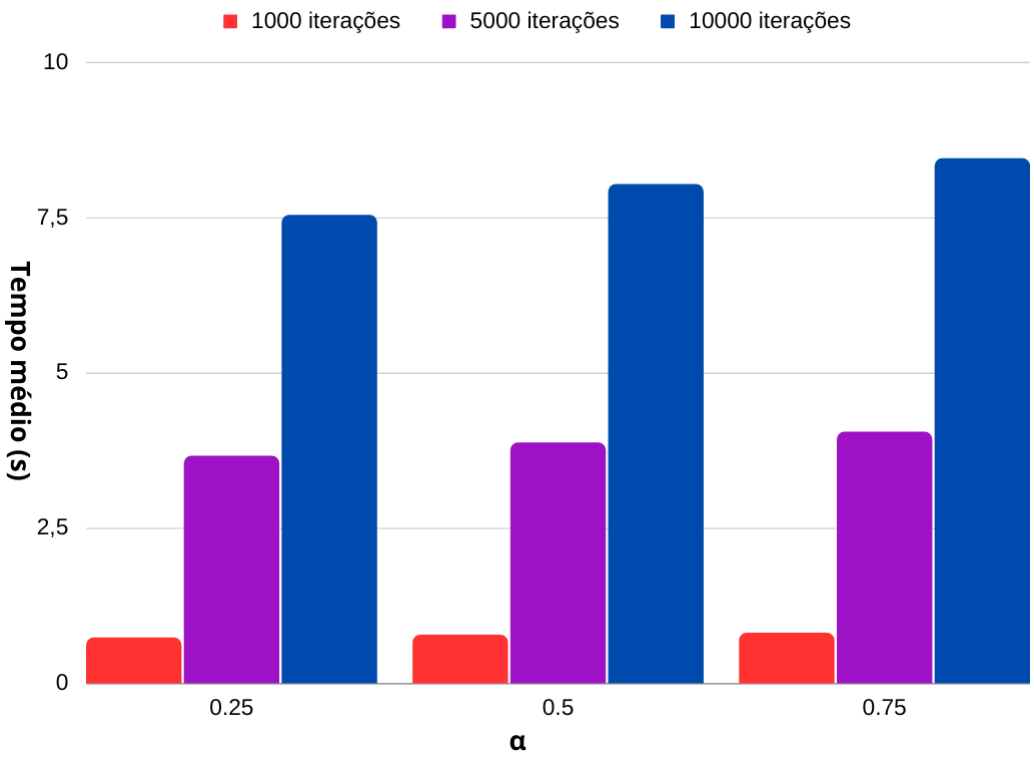


Figura 4.16: Tempos de execução obtidos para as instâncias com 10 recursos e 500 itens de ORLIB

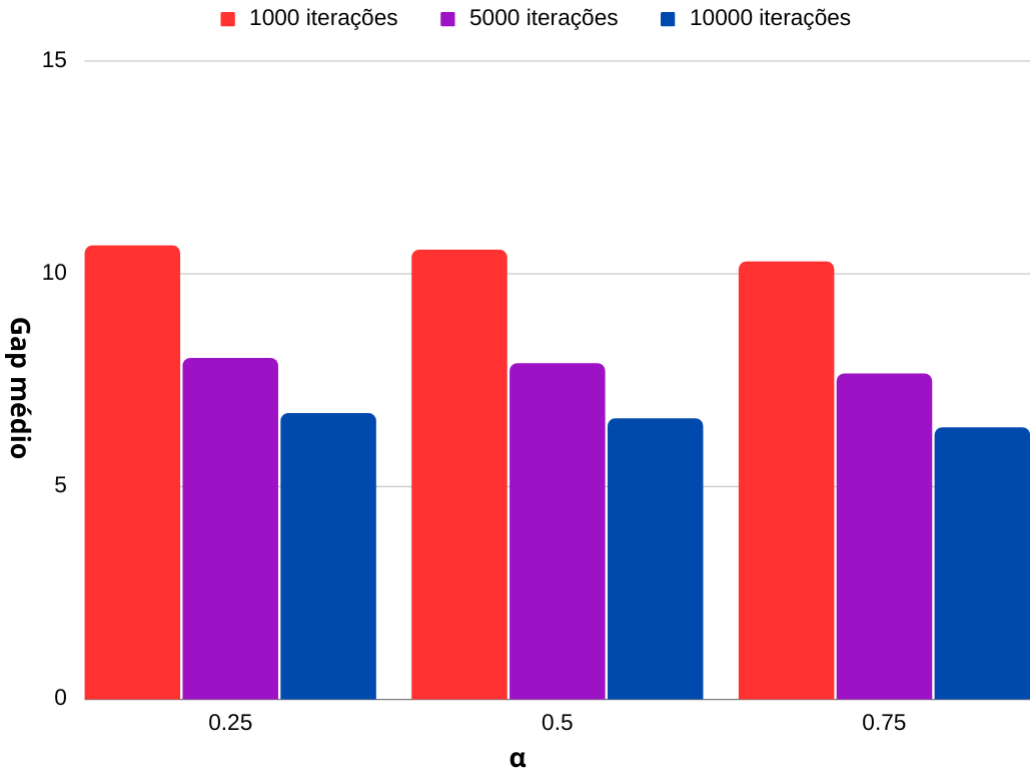


Figura 4.17: *Gaps* obtidos para as instâncias com 30 recursos e 100 itens de ORLIB

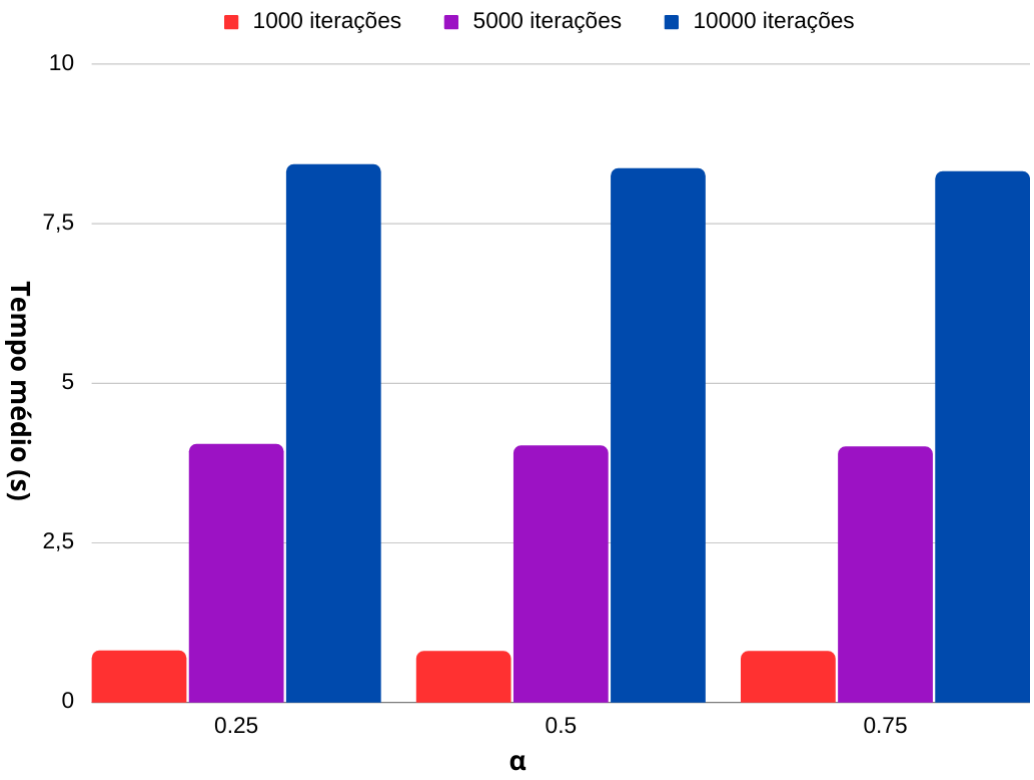


Figura 4.18: Tempos de execução obtidos para as instâncias com 30 recursos e 100 itens de ORLIB

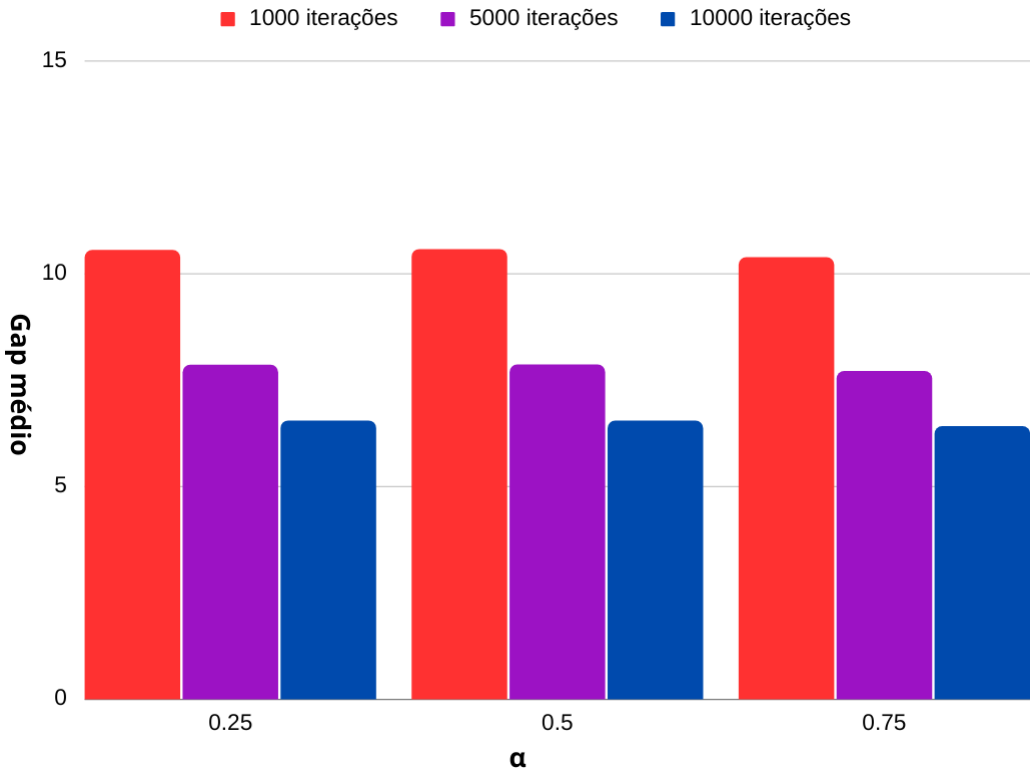


Figura 4.19: *Gaps* obtidos para as instâncias com 30 recursos e 250 itens de ORLIB

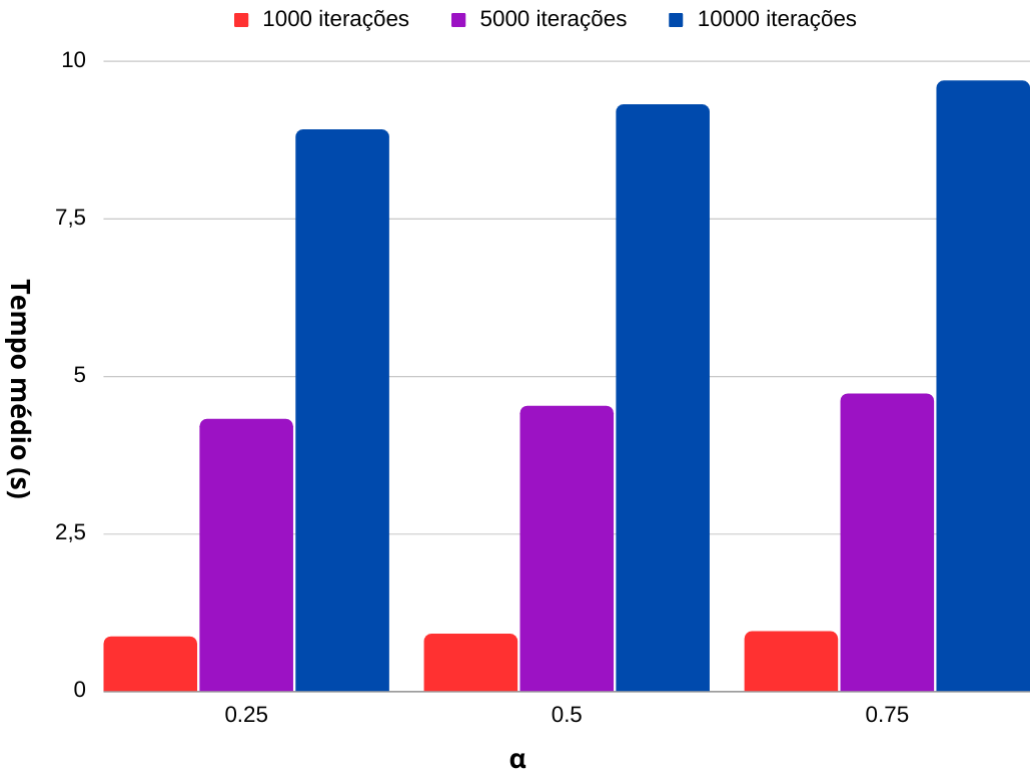


Figura 4.20: Tempos de execução obtidos para as instâncias com 30 recursos e 250 itens de ORLIB

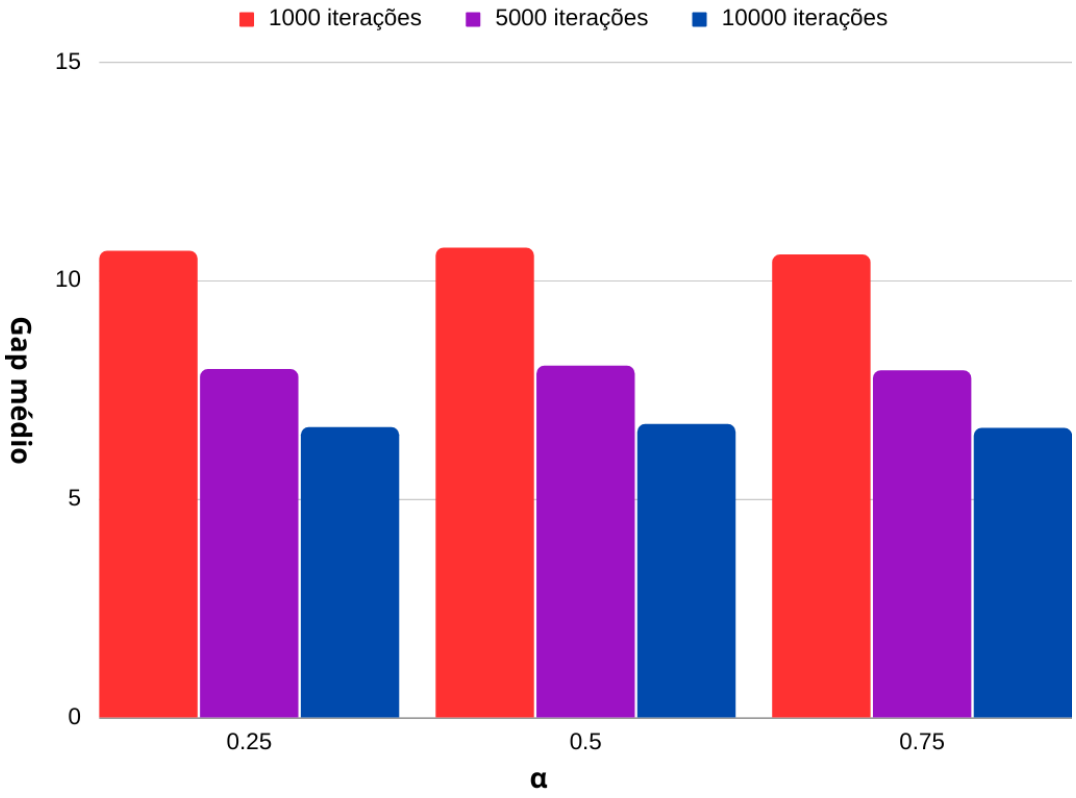


Figura 4.21: *Gaps* obtidos para as instâncias com 30 recursos e 500 itens de ORLIB

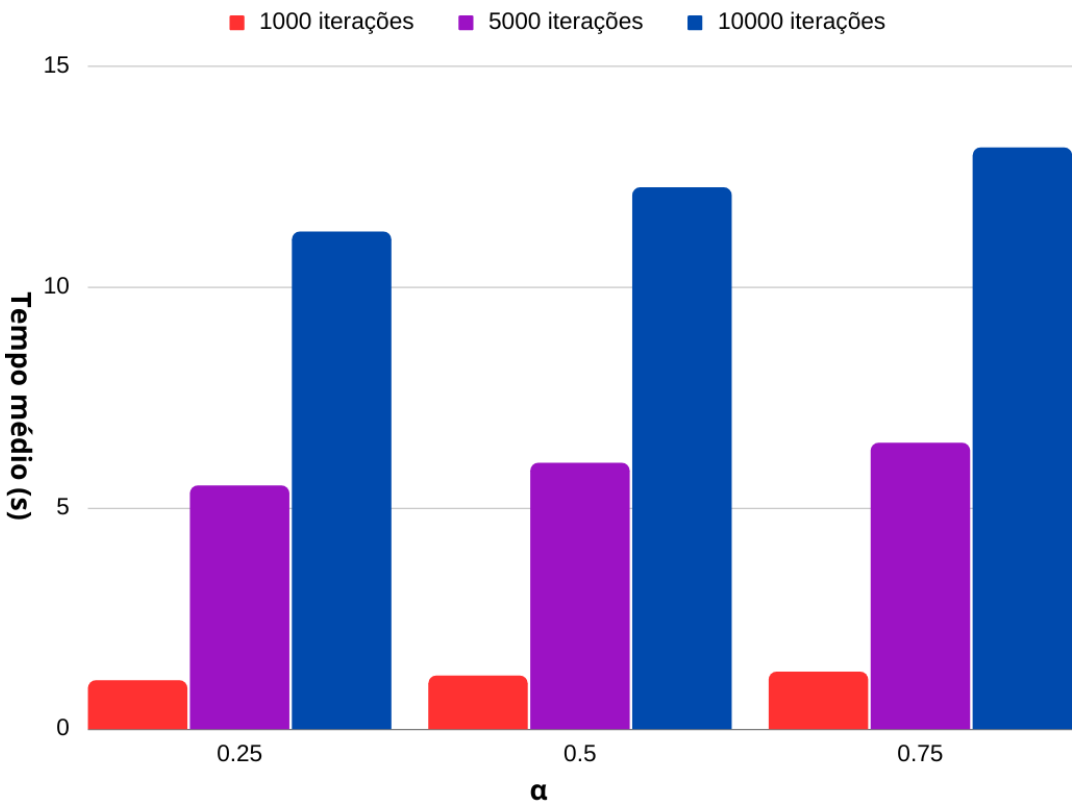


Figura 4.22: Tempos de execução obtidos para as instâncias com 30 recursos e 500 itens de ORLIB

Observando os gráficos anteriores, fica evidente o padrão mencionado anteriormente. Em todos os cenários avaliados, observou-se uma diminuição do *gap* à medida que o número de iterações aumentava, indicando uma melhoria na qualidade das soluções geradas pelo algoritmo. Em contrapartida, o tempo de execução aumentou à medida que o número de iterações cresceu. Vale ressaltar que houve casos significativos desse aumento, como pode ser visto no exemplo da instância gk11, ilustrado na Figura 4.4.

CAPÍTULO 5

Conclusão

Neste trabalho, foi explorada a aplicação do Algoritmo Colônia de Abelhas Artificiais (ABC - do inglês *Artificial Bee Colony*) na resolução do problema da mochila multidimensional binária (MKP - do inglês *Multidimensional Knapsack Problem*). O problema da mochila multidimensional binária é uma extensão do problema da mochila binária clássica, no qual, além de lidar com os pesos e os valores dos itens, existem várias dimensões e em cada uma delas uma restrição a ser respeitada.

Os resultados obtidos durante a realização deste estudo mostram que o ABC é uma abordagem promissora para lidar com problemas de otimização combinatória complexos, como o da mochila multidimensional binária. Durante o desenvolvimento deste estudo, foram realizados experimentos, utilizando instâncias de teste já abordadas na literatura, com diversas características, incluindo variações no número de itens, pesos individuais e capacidades.

Além disso, é importante destacar a importância da configuração correta dos parâmetros do algoritmo, como o número de abelhas, limite global, limite local e limite de ciclos. A escolha adequada desses parâmetros desempenhou um papel crucial na qualidade das soluções encontradas.

O ABC demonstrou um bom desempenho na otimização da função objetivo do MKP. Proporcionando para algumas instâncias, como hp e pb, soluções viáveis e até ótimas. Já para outras instâncias mais complexas, como GK e OR, foram encontradas soluções promissoras. Contudo, o algoritmo ainda possui margem para melhorias.

Nos experimentos realizados em diferentes conjuntos de dados de teste, utilizando do maior número de iterações testado, encontramos um *gap* médio de no máximo 8% para as instâncias mais desafiadoras.

Uma das principais vantagens do ABC é a sua natureza populacional, o que lhe permite explorar de maneira eficaz o espaço de busca e evitar a convergência prematura para mínimos locais. Isso o torna uma escolha interessante quando se lida com problemas de otimização combinatória complexos e de alta dimensionalidade, como o MKP.

Ainda assim, é importante ressaltar que o ABC não é uma solução universal e seus resultados podem variar dependendo das características específicas do problema. Por

exemplo, neste trabalho foi necessária a discretização do algoritmo a fim de utilizá-lo em um problema binário, já que originalmente o ABC foi projetado para problemas contínuos.

Para pesquisas futuras, sugere-se explorar ainda mais as capacidades do ABC no MKP, considerando testes em máquinas com maior poder computacional e um aumento do número de iterações. A paralelização do algoritmo pode ser uma abordagem interessante para melhorar o desempenho e o tempo de resposta. Além disso, a comparação do ABC com outros algoritmos de otimização populares pode fornecer um entendimento melhor sobre seu desempenho relativo em diferentes cenários.

Em síntese, esta pesquisa demonstrou a aplicabilidade do Algoritmo Colônia de Abelhas Artificiais na resolução do problema da mochila multidimensional Binária, oferecendo uma abordagem para enfrentar desafios em otimização combinatória. Com uma sintonização adequada e adaptação às características do problema, o ABC é uma ferramenta promissora para a comunidade de pesquisa em otimização.

Referências Bibliográficas

[1] ABDOLLAHZADEH, B., GHAREHCHOPOGH, F. S., MIRHALILI, S., "Artificial gorilla troops optimizer: A new nature-inspired metaheuristic algorithm for global optimization problems". International Journal of Intelligent Systems, 36: 5887-5958, 2021.

[2] [or]BEASLEY, J. E, "OR-Library: distributing test problems by electronic mail". Journal of the Operational Research Society 41.11, 1069–1072, 1990.

[3] CAMILLO, M., et al, "Combining exhaustive search and multi-objective evolutionary algorithm for service restoration in large-scale distribution systems". Electric Power Systems Research, 134, 1-8, 2016.

[4] DUARTE, G. R., "Um algoritmo inspirado em colônias de abelhas para otimização numérica com restrições". Dissertação Mestrado Acadêmico. Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas, 2015.

[5] DANTAS, B. A., "Metaheurísticas para o Problema da Mochila Multidimensional". Dissertação Doutorado. Universidade Federal de Mato Grosso do Sul, Faculdade de Computação, 2016.

[6] FINGLER, H., "Otimização de colônias de formigas em CUDA: o problema da mochila e o problema quadrático de alocação". Dissertação Mestrado Acadêmico. Universidade Federal de Mato Grosso do Sul, Faculdade de Computação, 2013.

[7] GLOVER, F., KOCHENBERGER, G., "Benchmarks for the multiple knapsack problem". abr de 2016. url: http://hces.bus.olemiss.edu/tools.html.

[8] KARABOGA, D., "An idea based on honey bee swarm for numerical optimization". Erciyes University, Engineering Faculty, 2005.

[9] KARABOGA, D., BASTURK, B., "A powerful and efficient algorithm for numerical function optiqmization: artificial bee colony (ABC) algorithm". Journal of Intelligent Manufacturing, 23, 4, 1001-1014, 2012.

[10] LAABADI, S., et al, "The 0/1 Multidimensional Knapsack Problem and Its Variants: A Survey of Practical Models and Heuristic Approaches". American Journal of Operations Research, 8, 395-439, 2018.

[11] SANTOS, K. D., MARCELOS, W. R., "Problemas de otimização (ou problemas de maximização e minimização)". Caderno de Resumos, Apresentação de Painel, 1, 3, 2015.

[12] SUNDAR, S., SINGH, A., ROSSI, R., "An Artificial Bee Colony Algorithm for the 0–1 Multidimensional Knapsack Problem". In: Ranka, S., et al, Contemporary Computing, 94, 141–15, Springer Berlin Heidelberg, 2010.

[13] VASCONCELOS, E. C. G. ., et al, "Ecological interactions in the foraging activity of Apis mellifera L. bees in an ecotone area in the municipality of Cocal-PI". Research, Society and Development, 10, 16, e333101623674, 2021.

[14] WEI, Y., "An improved binary artificial bee colony algorithm for solving multidimensional knapsack problem". International Conference on Electronic Information Engineering and Computer Technology (EIECT), 1208719, 1208719, 2021.

[15] ZHANG, S., LIU, S., "A Discrete Improved Artificial Bee Colony Algorithm for 0-1 Knapsack Problem". IEEE Access, 7, 104982-104991, 2019.